Recap

This lecture:

- Solving equations by Unification
- Matching and Unification algorithms
- Building-in axioms: $E$-Unification
Motivation

Unification: finding a common instance of two terms

Informally: we want to make two terms identical by finding the most general substitution of terms for variables.

Why?

- Applying rules in Isabelle: working out what \( ?P, ?Q, ?x \) are
- Heavily used in automated first-order theorem proving to postpone decisions during proof search: PROLOG, tableau provers, resolution provers
- Also used in most type inference algorithms (Haskell, OCaml, SML, Scala, ...)

A First Look at Unification

Unification: finding a common instance of two terms

Informally: we want to make two terms identical by finding the most general substitution of terms for variables.

Example
Can we make these pairs of terms equal by finding a common instance (assuming $X$, $Y$ are variables and $a$, $b$ are constants)?

<table>
<thead>
<tr>
<th>Term 1</th>
<th>Term 2</th>
<th>Yes/No</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(X, b)$ and $f(a, Y)$</td>
<td></td>
<td>Yes: $[a/X, b/Y]$</td>
<td>instance: $f(a, b)$</td>
</tr>
<tr>
<td>$f(X, X)$ and $f(a, b)$</td>
<td></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>$f(X, X)$ and $f(Y, g(Y))$</td>
<td></td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

Only (meta-)variables ($X$, $Y$, $Z$, …) can be replaced by other terms.
Matching

Problem

Given pattern and target find a substitution such that:

\[ \text{pattern}[\text{substitution}] \equiv \text{target} \]

where \( \equiv \) means that the terms are identical.

Example

\[(s(X) + Y)[0/X, s(0)/Y] \equiv (s(0) + s(0))\]

How we do find an adequate substitution?

We view matching as equation solving.
Matching (continued)

Discover a substitution by decomposing the equation to be solved along the term trees:

\[(s(X) + Y) \equiv (s(0) + s(0))\]
\[\downarrow\]
\[(s(X) \equiv s(0)) \land (Y \equiv s(0))\]
\[\downarrow\]
\[(X \equiv 0) \land (Y \equiv s(0))\]
### Some Abbreviations

<table>
<thead>
<tr>
<th>Term</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\overrightarrow{t}$</td>
<td>$t_1, \ldots, t_n \ (t \geq 1)$</td>
</tr>
<tr>
<td>$\bigwedge_i t_i$</td>
<td>$t_1 \land \ldots \land t_n$</td>
</tr>
<tr>
<td>vars($t$)</td>
<td>the set of free variables in $t$</td>
</tr>
<tr>
<td>Vars</td>
<td>the set of (all) free variables</td>
</tr>
</tbody>
</table>

vars($f(X, Y, g(a, Z, X))$) = \{X, Y, Z\}

vars($f(a, b, c)$) = \{\}
Matching as Equation Solving

Start with the *pattern* and *target* standardised apart:

\[ \text{vars}(\text{pattern}) \cap \text{vars}(\text{target}) = \{\} \]

Goal is to solve for \( \text{vars}(\text{pattern}) \) in equation \( \text{pattern} \equiv \text{target} \).

Strategy is to use transformation rules:

\[
\begin{align*}
\text{pattern} & \equiv \text{target} \\
\downarrow \\
\vdots \\
\downarrow \\
X_1 = t_1 \land \ldots \land X_n = t_n
\end{align*}
\]

Resulting substitution is \([t_1/X_1, \ldots, t_n/X_n]\).

Transformations end in failure if no match is possible.
Transformation Rules for Matching (Examples)

**Decompose**

\[ s(X) + Y \equiv s(0) + s(0) \]

\[ \Downarrow \]

\[ s(X) \equiv s(0) \land Y \equiv s(0) \]

**Conflict**

\[ s(X) + y \equiv s(0) \]

\[ \Downarrow \]

\[ \text{fail} \]

Cannot match: \( s \not\equiv + \)

**Eliminate**

\[ (X + Y \equiv s(0) + 0) \land (Y \equiv 0) \]

\[ \Downarrow \]

\[ (X + 0 \equiv s(0) + 0) \land (Y \equiv 0) \]

**Delete**

\[ X \equiv 0 \land (s(0) + 0 \equiv s(0) + 0) \]

\[ \Downarrow \]

\[ X \equiv 0 \]
Transformation Rules for Matching

**Assumptions:** $s$ and $t$ are arbitrary terms and are standardised apart.

<table>
<thead>
<tr>
<th>Name</th>
<th>Before</th>
<th>After</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decompose</td>
<td>$P \land f(\overrightarrow{s}) \equiv f(\overrightarrow{t})$</td>
<td>$P \land \bigwedge_i s_i \equiv t_i$</td>
<td></td>
</tr>
<tr>
<td>Conflict</td>
<td>$P \land f(\overrightarrow{s}) \equiv g(\overrightarrow{t})$</td>
<td>fail</td>
<td>$f \neq g$</td>
</tr>
<tr>
<td>Eliminate</td>
<td>$P \land X \equiv t$</td>
<td>$P[t/X] \land X \equiv t$</td>
<td>$X \in \text{vars}(P)$</td>
</tr>
<tr>
<td>Delete</td>
<td>$P \land t \equiv t$</td>
<td>$P$</td>
<td></td>
</tr>
</tbody>
</table>

Algorithm terminates when no further rules apply and fail has not occurred.

The algorithm terminates with a match iff there is one.

The algorithm may terminate without a match: e.g., $X = a \land b = Y$
Unification

Unification is two-way matching (there is no distinction between pattern and target).

\[ term_1[\text{substitution}] \equiv term_2[\text{substitution}] \]

Example

What substitution makes \((s(X) + s(0))\) and \((s(0) + Y)\) identical?

\[ \theta = \{0/X, s(0)/Y\} \]

We need to add extra rules to the matching algorithm:

\[
\begin{align*}
(s(X) + s(0)) & \equiv (s(0) + Y) \\
\downarrow & \quad \text{Decompose} \\
s(X) & \equiv s(0) \land s(0) \equiv Y \\
\downarrow & \quad \text{Decompose} \\
X & \equiv 0 \land s(0) \equiv Y \\
\downarrow & \quad \text{Switch} \\
X & \equiv 0 \land Y \equiv s(0)
\end{align*}
\]
# New Transformation Rules

## Switch

\[ t \equiv X \]

\[ \downarrow \]

\[ X \equiv t \]

Switch rule applies only if \( lhs \) is not originally a variable

## Coalesce

\[ X \equiv Y + 1 \land Y \equiv X \]

\[ \downarrow \]

\[ X \equiv X + 1 \land Y \equiv X \]

Similar to Eliminate, except both \( lhs \) and \( rhs \) are variables

## Occurs Check

\[ X \equiv X + 1 \]

\[ \downarrow \]

fail

\( lhs \) cannot occur in \( rhs \)

---

### Example

\[ f(X, X) \equiv f(Y, Y + 1) \]

\[ \downarrow \text{Decompose} \]

\[ X \equiv Y \land X \equiv Y + 1 \]

\[ \downarrow \text{Coalesce} \]

\[ X \equiv Y \land Y \equiv Y + 1 \]

\[ \downarrow \text{Occurs check} \]

fail

\[ p(X) \land X \equiv X + 1 \]

\[ \downarrow \text{Eliminate} \]

\[ p(X + 1) \land X \equiv X + 1 \]

\[ \downarrow \text{Eliminate} \]

\[ p((X + 1) + 1) \land X \equiv X + 1 \]

\[ \downarrow \text{Eliminate} \]

\[ \ldots \]

Non-termination can result without the occurs check.
## Unification Algorithm

**Assumptions:** \( s \) and \( t \) are arbitrary terms and \( \text{Vars} = \text{vars}(s) \cup \text{vars}(t) \).

<table>
<thead>
<tr>
<th>Name</th>
<th>Before</th>
<th>After</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decompose</td>
<td>( P \land f(s) \equiv f(t) )</td>
<td>( P \land \bigwedge_i s_i \equiv t_i )</td>
<td></td>
</tr>
<tr>
<td>Conflict</td>
<td>( P \land f(s) \equiv g(t) )</td>
<td>fail</td>
<td>( f \neq g )</td>
</tr>
<tr>
<td>Switch</td>
<td>( P \land s \equiv X )</td>
<td>( P \land X \equiv s )</td>
<td>( X \in \text{Vars} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( s \not\in \text{Vars} )</td>
</tr>
<tr>
<td>Delete</td>
<td>( P \land s \equiv s )</td>
<td>( P )</td>
<td></td>
</tr>
<tr>
<td>Eliminate</td>
<td>( P \land X \equiv s )</td>
<td>( P[s/X] \land X \equiv s )</td>
<td>( X \in \text{vars}(P) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( X \not\in \text{vars}(s) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( s \not\in \text{Vars} )</td>
</tr>
<tr>
<td>Occurs Check</td>
<td>( P \land X \equiv s )</td>
<td>fail</td>
<td>( X \in \text{vars}(s) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( s \not\in \text{Vars} )</td>
</tr>
<tr>
<td>Coalesce</td>
<td>( P \land X \equiv Y )</td>
<td>( P[Y/X] \land X \equiv Y )</td>
<td>( X, Y \in \text{vars}(P) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( X \neq Y )</td>
</tr>
</tbody>
</table>

- Conditions ensure that at most one rule applies to each conjunct
- Algorithm terminates with success when no further rules apply.
Composition of Unifiers (Substitutions)

Definition
If $\phi$ and $\theta$ are substitutions then their composition $\phi \circ \theta$ is also a substitution which, for any term $t$, satisfies the following property:

$$t[\phi \circ \theta] \equiv (t[\phi])[\theta]$$
Composition of Unifiers (Substitutions)

Definition
If $\phi$ and $\theta$ are substitutions then their *composition* $\phi \circ \theta$ is also a substitution which, for any term $t$, satisfies the following property:

$$t[\phi \circ \theta] \equiv (t[\phi])[\theta]$$

Examples:

$$[a/x] \circ [b/y] = [a/x, b/y]$$
Composition of Unifiers (Substitutions)

Definition
If \( \phi \) and \( \theta \) are substitutions then their composition \( \phi \circ \theta \) is also a substitution which, for any term \( t \), satisfies the following property:

\[
t[\phi \circ \theta] \equiv (t[\phi])[\theta]
\]

Examples:
\[
[a/x] \circ [b/y] = [a/x, b/y] \\
[g(y)/x] \circ [b/y] = [g(b)/x, b/y]
\]
Composition of Unifiers (Substitutions)

Definition
If \( \phi \) and \( \theta \) are substitutions then their composition \( \phi \circ \theta \) is also a substitution which, for any term \( t \), satisfies the following property:

\[
   t[\phi \circ \theta] \equiv (t[\phi])[\theta]
\]

Examples:
- \([a/x] \circ [b/y] = [a/x, b/y]\)
- \([g(y)/x] \circ [b/y] = [g(b)/x, b/y]\)
- \([a/x] \circ [b/x] = [a/x]\)
Composition of Unifiers (Substitutions)

Definition
If $\phi$ and $\theta$ are substitutions then their composition $\phi \circ \theta$ is also a substitution which, for any term $t$, satisfies the following property:

$$t[\phi \circ \theta] \equiv (t[\phi])[\theta]$$

Examples:

$$[a/x] \circ [b/y] = [a/x, b/y]$$
$$[g(y)/x] \circ [b/y] = [g(b)/x, b/y]$$
$$[a/x] \circ [b/x] = [a/x]$$

- Equality of substitutions: $\phi = \theta$ if $x[\phi] = x[\theta]$ for any variable $x$.
- Properties: $(\phi \circ \theta) \circ \sigma = \phi \circ (\theta \circ \sigma)$, $\phi \circ [] = \phi$ and $[] \circ \phi = \phi$.
- Composition is needed to define the notion of a most general unifier.
Properties of the Unification Algorithm

- The algorithm will find a unifier, if it exists.
- It returns the **most general unifier** (mgu) $\theta$.

**Definition**

Given any two terms $s$ and $t$, $\theta$ is their mgu if:

$$s[\theta] \equiv t[\theta] \land \forall \phi. \ s[\phi] \equiv t[\phi] \rightarrow \exists \psi. \ \phi = \theta \circ \psi.$$

Consider $g(g(X))$ and $g(Y)$. Is $[g(3)/Y, 3/X]$ a unifier? Is it the mgu?

- mgu is **unique** up to alphabetic variance;
- the algorithm can easily be extended to simultaneous unification on $n$ expressions.
Building-in Axioms

General Scheme:

\[(Ax_1 \cup Ax_2) + unif \implies Ax_1 + unif_{Ax_2}.\]

Some axioms of the theory become built into unification.

Example

Commutative-Unification

\[X + 2 = Y + 3\]

\[\downarrow\]

\[Y = 2 \land X = 3\]

We no longer use \(\equiv\) but =

How do we deal with this?
We can add a new transformation rule (Mutate rule).
### Unification Algorithm for Commutativity

<table>
<thead>
<tr>
<th>Name</th>
<th>Before</th>
<th>After</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decompose</td>
<td>$P \land f(\vec{s}) = f(\vec{t})$</td>
<td>$P \land \bigwedge_i s_i = t_i$</td>
<td></td>
</tr>
<tr>
<td>Conflict</td>
<td>$P \land f(\vec{s}) = g(\vec{t})$</td>
<td>fail</td>
<td>$f \neq g$</td>
</tr>
<tr>
<td>Switch</td>
<td>$P \land s = X$</td>
<td>$P \land X = s$</td>
<td>$X \in Vars$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$s \not\in Vars$</td>
</tr>
<tr>
<td>Delete</td>
<td>$P \land s = s$</td>
<td>$P$</td>
<td></td>
</tr>
<tr>
<td>Eliminate</td>
<td>$P \land X = s$</td>
<td>$P[s/X] \land X = s$</td>
<td>$X \in \text{vars}(P)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$X \not\in \text{vars}(s)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$s \not\in Vars$</td>
</tr>
<tr>
<td>Check</td>
<td>$P \land X = s$</td>
<td>fail</td>
<td>$X \in \text{vars}(s)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$s \not\in Vars$</td>
</tr>
<tr>
<td>Coalesce</td>
<td>$P \land X = Y$</td>
<td>$P[Y/X] \land X = Y$</td>
<td>$X, Y \in \text{vars}(P)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$X \neq Y$</td>
</tr>
<tr>
<td>Mutate</td>
<td>$P \land f(s_1, t_1) = f(s_2, t_2)$</td>
<td>$P \land s_1 = t_2 \land t_1 = s_2$</td>
<td>$f$ is commutative</td>
</tr>
</tbody>
</table>

Decompose and Mutate rules overlap.
Most General Unifiers

For ordinary unification, the mgu is unique, but what happens when new rules are built-into the unification algorithm?

Multiple mgus: Commutative unification

\[ X + Y = a + b \rightarrow \begin{cases} X = a & Y = b \\ X = b & Y = a \end{cases} \quad \text{Both are equally general.} \]

Infinitely many mgus: Associative unification \( X + (Y + Z) = (X + Y) + Z \).

\[ X + a = a + X \rightarrow \begin{cases} X = a \\ X = a + a \\ X = a + a + a \vdots \end{cases} \quad \text{All independent (not unifiable).} \]

No mgus: Build in \( f(0, X) = X \) and \( g(f(X, Y)) = g(Y) \):

\[ g(X) = g(a) \rightarrow \begin{cases} X = a \\ X = f(Y_1, a) \\ X = f(Y_1, f(Y_2, a)) \end{cases} \quad \text{Many unifiers but no mgu.} \]
Types of Unification

**Unitary**  A single unique mgu, or none (predicate logic).

**Finitary**  Finite number of mgus (predicate logic with commutativity).

**Infinitary**  Possibly infinite number of mgus (predicate logic with associativity).

**Nullary**  No mgus exist, although unifiers may exist.

**Undecidable**  Unification not decidable — no algorithm.
## Types of Unification

<table>
<thead>
<tr>
<th>Axioms</th>
<th>Type</th>
<th>Decidable</th>
</tr>
</thead>
<tbody>
<tr>
<td>nil</td>
<td>unitary</td>
<td>yes</td>
</tr>
<tr>
<td>commutative</td>
<td>finitary</td>
<td>yes</td>
</tr>
<tr>
<td>associative</td>
<td>infinitary</td>
<td>yes</td>
</tr>
<tr>
<td>assoc. + dist.</td>
<td>infinitary</td>
<td>yes</td>
</tr>
<tr>
<td>lambda calculus</td>
<td>infinitary</td>
<td>no</td>
</tr>
<tr>
<td>$\lambda$-calculus pattern fragment</td>
<td>unitary</td>
<td>yes</td>
</tr>
</tbody>
</table>
Summary

- Unification (Bundy Ch. 17.1 - 17.4)
  - Algorithms for matching and unification.
  - Unification as equation solving.
  - Transformation rules for equation solving.
  - Building-in axioms. (E-Unification/Semantic Unification)
  - Most general unifiers and classification.

- Next time: Proof by rewriting